Numerical solution of ordinary differential equations

Anne Kværnø

Nov 15, 2022

1 Error estimation and step size control

To control the global error $y(t_n) - y_n$ is notoriously difficult, and far beyond what will be discussed in this course. To control the local error in each step and adjust the step size accordingly is rather straightforward, as we will see.

1.1 Error estimation

Given two methods, one of order p and the other of order $p+1$ or higher. Assume we have reached a point $(t_n, \mathbf{y}_n)$. One step forward with each of these methods can be written as

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\boldsymbol{\Phi}(t_n, \mathbf{y}_n; h), && \text{order } p, \\ \widehat{\mathbf{y}}_{n+1} &= \mathbf{y}_n + h\widehat{\boldsymbol{\Phi}}(t_n, \mathbf{y}_n; h), && \text{order } p+1 \text{ or more.} \end{aligned}$$

Let $\mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n)$ be the exact solution of the ODE through $(t_n, \mathbf{y}_n)$. We would like to find an estimate for *the local error* $\mathbf{l}_{n+1}$, that is, the error in one step starting from $(t_n, \mathbf{y}_n)$,

$$\mathbf{l}_{n+1} = \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1}.$$

As we already have seen, the local error is found by finding the power series in h of the exact and the numerical solution. The local error is of order p if the lowest order terms in the series where the exact and the numerical solution differ is of order $p+1$. So the local errors of the two methods are

$$\begin{aligned} \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1} &= \boldsymbol{\Psi}(t_n, \mathbf{y}_n)h^{p+1} + \ldots, \\ \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \widehat{\mathbf{y}}_{n+1} &= \qquad\qquad + \ldots, \end{aligned}$$

where $\boldsymbol{\Psi}(t_n, \mathbf{y}_n)$ is a term consisting of method parameters and differentials of $\mathbf{f}$ and $\ldots$ contains all the terms of the series of order $p+2$ or higher. Taking the difference gives

$$\widehat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1} = \boldsymbol{\Psi}(t_n, \mathbf{y}_n)h^{p+1} + \ldots.$$

Assume now that h is small, such that the *principal error term* $\boldsymbol{\Psi}(t_n, \mathbf{y}_n)h^{p+1}$ dominates the error series. Then a reasonable approximation to the unknown local error $\mathbf{l}_{n+1}$ is the *local error estimate* $\mathbf{le}_{n+1}$:

$$\mathbf{le}_{n+1} = \widehat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1} \approx \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1}.$$

Example: Apply Euler's method of order 1 and Heun's method of order 2 with $h = 0.1$ to the equation

$$y' = -2ty, \qquad y(0) = 1.$$

Use this to find an approximation to the error after one step.

Euler's method:

$$y_1 = 1.0 - 0.1 \cdot 2 \cdot 0 \cdot 1.0 = 1.0.$$

Heun's method

$$\begin{aligned} k_1 &= -2 \cdot 0.0 \cdot 1.0 = 0.0, \\ k_2 &= -2 \cdot 0.1 \cdot (1 + 0.0) = -0.2, \\ \widehat{y}_1 &= 1.0 + \frac{0.1}{2} \cdot (0.0 - 0.2) = 0.99. \end{aligned}$$

The error estimate and the local error are respectively

$$le_1 = \widehat{y}_1 - y_1 = -10^{-2}, \qquad l_1 = y(0.1) - y_1 = e^{-0.1^2} - 1.0 = -0.995 \cdot 10^{-2}.$$

So in this case the error estimate is a quite decent approximation to the actual local error.

1.2 Stepsize control

The next step is to control the local error, that is, choose the step size so that $\|\mathbf{le}_{n+1}\| \leq \text{Tol}$ for some given tolerance Tol, and for some chosen norm $\|\cdot\|$.

Essentially:

Given $t_n, \mathbf{y}_n$ and a step size h_n.

- Perform one step with the method of choice, and find the error estimate $\mathbf{le}_{n+1}$.
- if $\|\mathbf{le}\|_{n+1} < \text{Tol}$

  Accept the solution $t_{n+1}, \mathbf{y}_{n+1}$.

  If possible, increase the step size for the next step.

- else

  Repeat the step from $(t_n, \mathbf{y}_n)$ with a reduced step size h_n.

In both cases, the step size will change. But how?

From the discussion above, we have that

$$\|\mathbf{le}_{n+1}\| \approx D h_n^{p+1}.$$

where $\mathbf{le}_{n+1}$ is the error estimate we can compute, D is some unknown quantity, which we assume almost constant from one step to the next. We aim for a step size h_{new} such that

$$\text{Tol} \approx D h_{\text{new}}^{p+1}.$$

From these two approximations we get:

$$\frac{\text{Tol}}{\|\mathbf{le}_{n+1}\|} \approx \left(\frac{h_{\text{new}}}{h_n}\right)^{p+1} \qquad \Rightarrow \qquad h_{\text{new}} \approx \left(\frac{\text{Tol}}{\|\mathbf{le}_{n+1}\|}\right)^{\frac{1}{p+1}} h_n.$$

To avoid too many rejected steps, it is wise to be a bit conservative when choosing the new step size, so the following is used in practice:

$$h_{\text{new}} = P \cdot \left(\frac{\text{Tol}}{\|\mathbf{le}_{n+1}\|}\right)^{\frac{1}{p+1}} h_n.$$

where the *pessimist factor* $P < 1$ is some constant, normally chosen between 0.5 and 0.95.

1.3 Implementation

We have all the bits and pieces for constructing an adaptive ODE solver based on Euler's and Heuns's methods. There are still some practical aspects to consider:

- The combination of the two methods, implemented in `heun_euler` can be written as

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n), \\
\mathbf{k}_2 &= \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_1), \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + h\mathbf{k}_1, && \text{Euler} \\
\widehat{\mathbf{y}}_{n+1} &= \mathbf{y}_n + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2), && \text{Heun} \\
\mathbf{le}_{n+1} &= \|\widehat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}\| = \frac{h}{2}\|\mathbf{k}_2 - \mathbf{k}_1\|.
\end{aligned}$$

- Even if the error estimate is derived for the lower order method, in this case Euler's method, it is common to advance the solution with the higher order method, since the additional accuracy is for free. This is usually referred to as *local extrapolation*.
- Adjust the last step to be able to terminate the solutions exactly in t_{end}.
- To avoid infinite loops, add some stopping criteria. In the code below, there is a maximum number of allowed steps (rejected or accepted).
- The main driver `ode_adaptive` is written to make it simple to test other pairs of methods. This is also the reason why the function `heun_euler` returns the order of the lowest order method.

Numerical example: Apply the code to the test equation:

$$y' = -2ty, \qquad y(0) = 1.$$

using $\text{Tol} = 10^{-3}$ and $h_0 = 0.1$.

See the function `ode_example4()` in `numode.py`.

Numerical exercises:

1. Solve the Lotka-Volterra equation, use for instance $h_0 = 0.1$ and $\text{Tol} = 10^{-3}$. Notice also how the step size varies over the integration interval.
2. Repeat the experiment using Van der Pol's equation.

A Runge–Kutta methods with an error estimate are usually called *embedded Runge–Kutta methods* or *Runge–Kutta pairs*, and the coefficients can be written in a Butcher tableau as follows

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}	
c_2	a_{21}	a_{22}	$\cdots$	a_{2s}	
$\vdots$	$\vdots$			$\vdots$	
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss}	
	b_1	b_2	$\cdots$	b_s	Order p
	$\widehat{b}_1$	$\widehat{b}_2$	$\cdots$	$\widehat{b}_s$	Order $p+1$

The error estimate is then given by

$$\mathbf{le}_{n+1} = h\sum_{i=1}^{s}(\widehat{b}_i - b_i)\mathbf{k}_i.$$

Using this notation, the Heun-Euler pair is presented as

0		
1	1	
	1	0
	$\frac{1}{2}$	$\frac{1}{2}$

2 Stiff ODEs

Let us start this section by an example:

Numerical example 1s: Given the following system of two ODEs

$$\begin{aligned} y_1' &= -2y_1 + y_2 + 2\sin(t), & y_1(0) &= 2, \\ y_2' &= (a-1)y_1 - ay_2 + a\big(\cos(t) - \sin(t)\big), & y_2(0) &= 3, \end{aligned}$$

where a is some positive parameter. The exact solution, which is independent of the parameter, is

$$y_1(t) = 2e^{-t} + \sin(t), \qquad y_2(t) = 2e^{-t} + \cos(t).$$

Solve this problem now with some adaptive ODE solver, for instance the Heun-Euler scheme.

Now try the tolerances Tol $= 10^{-2}$, 10^{-4}, 10^{-6}, and perform the experiment with two different values of the parameters, $a = 2$ and $a = 999$.

See `ode_example_1s` in `numode.py`.

For $a = 2$ the expected behaviour is observed, for higher accuracy, more steps are required. But the example $a = 999$ requires much more steps, and the step size seems almost independent of the tolerance, at least for Tol $= 10^{-2}$, 10^{-4}.

The example above with $a = 999$ is a typically example of a *stiff ODE*. What defines these types of ODEs is that there are (at least) two different time scales at play at the same time: a slow time scale that dominates the time evolution of the solution of the ODE, and a fast time scale at which small perturbations of the solution may occur. In physical systems, this might be due to very strong damping of selected components of the system.

Numerical example 1, cont.: The general solution of the ODE can be shown to be

$$\mathbf{y}(t) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-t} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)t} + \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$$

for some constants c_1, c_2. The terms e^{-t}, $\sin(t)$, and $\cos(t)$ evolve at a time scale of order 1. In contrast, the term $e^{-(a+1)t}$ reverts back to being essentially equal to zero at a time scale of order $1/(a+1)$.

When a stiff ODE is solved by some explicit adaptive method like the Heun-Euler scheme, an unreasonably large number of steps is required, and this number seems independent of the tolerance. The problem is that, for explicit methods, the local error is dominated by what is happening at the fast time scale, and the step length will be adapted to that time scale as well. Even worse, any larger step size will lead to instabilities and exponentially increasing oscillations in the numerical solution.

In the remaining part of this note we will explain why this happens, and how we can overcome the problem. For simplicity, the discussion is restricted to linear problems, but also nonlinear ODEs can be stiff, and often will be.

Exercise s1: Repeat the experiment on the Van der Pol equation

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2, \\ y_2' &= \mu(1 - y_1^2)y_2 - y_1, & y_2(0) &= 0. \end{aligned}$$

Use $\mu = 2$, $\mu = 5$ and $\mu = 50$.

2.1 Linear stability analysis

The phenomenon observed above can be analyzed by studying the very simple linear test equation

$$y' = \lambda y, \qquad y(0) = y_0,$$

where the parameter $\lambda \in \mathbb{C}$ satisfies

$$\Re\lambda < 0.$$

and represent the fastest decaying component in a system. A more extensive explanation will be given below. Here, and in the following, $\Re\lambda$ will denote the real part of λ, and $\Im\lambda$ will denote the imaginary part of λ.

The analytic solution of this problem is

$$y(x) = y_0\, e^{\lambda x} = y_0\, e^{\Re\lambda\, x}\big(\cos(\Im\lambda\, x) + i\sin(\Im\lambda\, x)\big).$$

Since $\Re\lambda < 0$, the solution $y(x)$ tends to zero as $x \to \infty$. We want a similar behaviour for the numerical solution, that is $|y_n| \to 0$ when $n \to \infty$.

One step of some Runge–Kutta method applied to the linear test equation can always be written as

$$y_{n+1} = R(z)y_n, \qquad z = \lambda h.$$

The function $R(z)\colon \mathbb{C} \to \mathbb{C}$ is called the *stability function* of the method. $R(z)$ is either a polynomial or a rational function of z.

Example 2s: The application of Euler's method for the linear test equation results in the sequence

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n = (1 + z)y_n \qquad \text{with } z = h\lambda.$$

The stability function of Euler's method is therefore the function

$$R(z) = 1 + z.$$

For a comparison, Heun's method for this test equation is

$$\begin{aligned} k_1 &= \lambda y_n, \\ k_2 &= \lambda(y_n + hk_1), \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2), \end{aligned}$$

which can be rewritten as

$$y_{n+1} = y_n + \frac{h}{2}(\lambda y_n + \lambda(y_n + h\lambda y_n) = y_n + h\lambda y_n + \frac{(h\lambda)^2}{2}y_n.$$

As a consequence, the stability function for Heun's method is

$$R(z) = 1 + z + \frac{z^2}{2}.$$

One step with the Trapezoidal rule is

$$y_{n+1} = y_n + h\frac{1}{2}(\lambda y_n + \lambda y_{n+1})$$

and the corresponding stability function is

$$R(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}$$

We now return back to the analysis of the behaviour of an arbitrary Runge-Kutta method with stability function R. Taking the absolute value on each side of the expression

$$y_{n+1} = R(z)y_n,$$

we see that there are three possible outcomes:

$$\begin{array}{llllll} |R(z)|<1 & \Rightarrow & |y_{n+1}|<|y_n| & \Rightarrow & y_n \to 0 & \text{(stable)} \\ |R(z)|=1 & \Rightarrow & |y_{n+1}|=|y_n| & & & \\ |R(z)|>1 & \Rightarrow & |y_{n+1}|>|y_n| & \Rightarrow & |y_n| \to \infty & \text{(unstable)} \end{array}$$

The *stability region* of a method is defined by

$$\mathcal{S} = \{z \in \mathbb{C} \ : \ |R(z)| \leq 1\}.$$

To get a stable numerical solution, we have to choose the step size h such that $z = \lambda h \in \mathcal{S}$.

Example 2s, continued: In the case of Euler's method, we have obtained the stability function

$$R(z) = 1 + z.$$

As a consequence, the stability region for Euler's method is

$$\mathcal{S} = \{z \in \mathbb{C} \ : \ |1 + z| \leq 1\}.$$

This is a ball in the complex plane, which is centered at -1 and has a radius of 1.

Linear systems of ODEs. We are given a system of m differential equation of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(x). \qquad (*)$$

Such systems have been discussed in Mathematics 3, and the technique for finding the exact solution will shortly be repeated here:

Solve the homogenous system $\mathbf{y}' = A\mathbf{y}$, that is, find the eigenvalues λ_i and the corresponding eigenvectors $\mathbf{v}_i$ satisfying

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad i = 1, 2, \ldots, m. \qquad (**)$$

Assume that A has a full set of linearly independent (complex) eigenvectors $\mathbf{v}_i$ with corresponding (complex) eigenvalues λ_i. Let $V = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$, and $\Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_m\}$. Then V is invertible and

$$AV = V\Lambda \qquad \text{and therefore} \qquad V^{-1}AV = \Lambda.$$

The ODE (*) can thus be rewritten as

$$V^{-1}\mathbf{y}' = V^{-1}AVV^{-1}\mathbf{y} + V^{-1}\mathbf{g}(t).$$

Let $\mathbf{z} = V^{-1}\mathbf{y}$ and $\mathbf{q}(t) = V^{-1}\mathbf{g}(t)$ such that the equation can be decoupled into a set of independent scalar differential equations

$$\mathbf{z}' = \Lambda\mathbf{z} + \mathbf{q}(t) \qquad \text{or, equivalently} \qquad z_i' = \lambda_i z_i + q_i(t), \quad i = 1, \ldots, m.$$

The solution of such equations has been discussed in Mathematics 1. When these solutions are found, the exact solution is given by

$$\mathbf{y}(t) = V\mathbf{z}(t),$$

and possible integration constants are given by the initial values.

As it turns out, the eigenvalues $\lambda_i \in \mathbb{C}$ are the key to understanding the behaviour of the ODE integrators, and it motivates the study of the stability properties of the very simplified, though complex, linear test equation

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}^-.$$

The discussion below is also relevant for nonlinear ODEs $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$, in which case we have to consider the eigenvalues of the Jacobian $\mathbf{f}_\mathbf{y}$ of $\mathbf{f}$ with respect to $\mathbf{y}$.

Example 1: We now return to the introductory example. There, the ODE can be written as

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t),$$

with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \qquad \mathbf{g}(t) = \begin{pmatrix} \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix}.$$

The eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(a+1)$. The general solution is given by

$$\mathbf{y}(t) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-t} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)t} + \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}.$$

In the introductory example, the initial values were chosen such that $c_1 = 2$ and $c_2 = 0$. However, for large values of a, the term $e^{-(a+1)t}$ will still go to 0 almost immediately, even if $c_2 \neq 0$. It is this term that creates problems for the numerical solution.

Numerical example 2: We have already discussed the stability function and stability region for Euler's method in the example above. We now solve the introductory problem

$$\mathbf{y}' = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix} \mathbf{y} + \begin{pmatrix} \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix}, \qquad \mathbf{y}(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \qquad a > 0.$$

by Euler's method. We know that the eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(1+a)$.

For the numerical solution to be stable for both eigenvalues, we have to require that the step length h satisfies

$$|1 + \lambda_1 h| \leq 1 \qquad \text{and} \qquad |1 + \lambda_2 h| \leq 1.$$

Since both eigenvalues in this case are real and negative, we see after a short computation that this results in the requirement that

$$h \leq \frac{2}{1+a}.$$

Try $a = 9$ and $a = 999$. Choose step sizes a little bit over and under the stability boundary, and you can experience that the result is sharp. If h is just a tiny bit above, you may have to increase the interval of integration to see the unstable solution.

It is the term corresponding to the eigenvalue $\lambda_2 = -(a+1)$ which makes the solution unstable. And the solution oscillate since $R(z) < -1$ for $h > 2/(1+a)$.

Exercise 2:

1. Find the stability region for Heun's method.
2. Repeat the experiment in Example 2 using Heun's mehod.

NB! Usually the error estimation in adaptive methods will detect the unstability and force the step size to stay inside or near the stability region. This explains the behaviour of the experiment in the introduction of this note.

3 A-stable methods.

In an ideal world, we would prefer the stability region to satisfy

$$\mathcal{S} \supset \mathbb{C}^- := \{z \in \mathbb{C} \ : \ \Re z \leq 0\},$$

such that the method is stable for all $\lambda \in \mathbb{C}$ with $\Re\lambda \leq 0$ and for all h. Such methods are called A-stable. For all explicit methods, like Euler's and Heun's, the stability function will be a polynomial, and $|R(z)| \to \infty$ as $\Re z \to -\infty$. Explicit methods can never be A-stable, and we therefore have to search among implicit methods. The simplest of those is the implicit, or backward, Euler's method, given by

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

Applied to the linear test equation $y' = \lambda y$, this results in the update

$$y_{n+1} = y_n + h\lambda y_{n+1} \qquad \text{or} \qquad y_{n+1} = \frac{1}{1-h\lambda} y_n.$$

We therefore see that we have the stability function

$$R(z) = \frac{1}{1-z}.$$

The stability region for the implicit Euler function is thus

$$\mathcal{S} = \left\{ z \in \mathbb{C} \ : \ \left| \frac{1}{1-z} \right| \leq 1 \right\} = \{ z \in \mathbb{C} \ : \ |1-z| \geq 1 \}.$$

This is the whole complex plan apart from an open ball with center $+1$ and radius 1. Thus the method is A-stable, as every complex number z with $\Re z \leq 0$ is contained in $\mathcal{S}$.

3.1 Implementation of implicit Euler's method

For simplicity, we will only discuss the implementation of implicit Euler's method for linear systems of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t),$$

where A is a constant matrix. In this case, one step of implicit Euler is given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hA\mathbf{y}_{n+1} + h\mathbf{g}(t_{n+1}).$$

Thus a linear system

$$(I - hA)\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{g}(t_{n+1})$$

has to be solved with respect to $\mathbf{y}_{n+1}$ for each step.

In the implementation below, the right hand side of the ODE is implemented as a function `rhs`, returning the matrix A and the vector $\mathbf{g}(x)$ for each step. The function `implicit_euler` performs one step with implicit Euler. It has the same interface as the explicit method, so that the function `ode_solve` can be used as before.

Numerical example 3: Solve the test equation with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \qquad \mathbf{g}(t) = \begin{pmatrix} \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix},$$

by the implicit Euler method. Choose $a = 2$ and $a = 999$, and try different stepsizes like $h = 0.1$ and $h = 0.01$. Are there any stability issues in this case?

Exercise 2: The trapezoidal rule is an implicit method which for a general ODE $\mathbf{y}'(x) = f(x, \mathbf{y}(x))$ is given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}\Big(\mathbf{f}(x_n, \mathbf{y}_n) + \mathbf{f}(x_{n+1}, \mathbf{y}_{n+1}) \Big).$$

1. Find the stability function to the trapezoidal rule, and prove that it is A-stable.
2. Implement the method, and repeat the experiment above.

3.2 Adaptive methods.

Implicit Euler is a method of order 1, and the trapezoidal rule of order 2. Thus, these can be used for error estimation: Perform one step with each of the methods, use the difference between the solutions as an error estimate, and use the solution from the trapezoidal rule to advance the solution. This has been implemented in the function `trapezoidal_ieuler`. The interface is as for the embedded pair `heun_euler`, so the adaptive solver `ode_adaptive` can be used as before.

Numerical example 4: Repeat the experiment from the introduction, using `trapezoidal_ieuler`.

We observe that there are no longer any step size restriction because of stability. The algorithm behaves as expected.

Comment: Implicit methods can of course also be applied for nonlinear ODEs. Implicit Euler's method will be

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x, \mathbf{y}_{n+1}),$$

which is a nonlinear system which has to be solved for each step. Similar for the trapezoidal rule. Usually these equations are solved by Newton's method or some simplification of it.